Compte rendu de projet
Info 0402

Morvan LASSAUZAY

Sacha KIERBEL

Sommaire

Présentation du projet et analyse des besoins — 0

- Le jeu d'échec — 1
- Les besoins — 1

Solution technique et interaction entre les composants — 4

- Définition des composants — 4
- Interaction entre les composants — 8

Bilan — 10

- Manuel d'utilisation — 10
- Conclusion personnelle — 11

Remerciements — 13

I-Présentation du projet et analyse des besoins.

A) Le jeu d'échec.

Il s'agit d'un jeu introduit en Europe dans le courant du X ème siècle et dont les règles encore usées de nos jours ont été introduites dès la fin du XV ème. Également désigné par l'appelation «Les échecs» , il fait s'opposer deux joueurs sur un plateau quadrillé de 64 cases blanches et noires. L'un après l'autre les joueurs vont mouvoir, selon le(s) déplacement(s) spécifique(s) propre(s) à chacune d'entre elle, une pièce de leur camp. Chaque camp est composé de huit pions, deux fous, deux cavaliers, deux tours, une reine et un roi. L'objectif d'une partie est de mettre l'adversaire en situation «d'échec et mat», c'est à dire de coincer le Roi adverse de telle sorte qu'il ne puisse se soustraire à son élimination.

Tranposer ce jeu informatiquement avec le langage C++ représente donc un exercice intéressant. En effet, les échecs dans leur conception vont devoir faire appel à de nombreuses notions de programmation et de mathématiques : héritage, matrice, création de nouveaux types, encapsulation, gestion de la mémoire... Pour respecter au mieux les règles du jeu et pour y ajouter les avantages/contraintes d'une version virtuelle (possibilité de sauvegarde de partie, gestion du tour par tour, des pièces prises etc...) il va donc être nécessaire d'organiser rigoureusement son code et d'utiliser toutes les ressources du C++ afin de satisfaire une représentation correcte du jeu d'Échecs.

B) Les besoins.

Le jeu d'échec est, comme son nom l'indique, un jeu. Qu'est-ce qui définit un jeu ? En quelque sorte, ce sont ses règles. Que sont les règles ? Ce sont les contraintes imposées au joueur qui vont limiter ses possibilités d'action pour instaurer une certaine difficulté et une ligne de conduite caractérisant le jeu. Ce sont ces contraintes qu'il nous incombe d'assurer afin d'être fidèle au jeu et de le reproduire convenablement. Concrètement cela signifie qu'une fois l'environnement établi sur l'ordinateur (en l'occurence il s'agit du plateau et des pièces), il faut contrôler le tour à tour des joueurs, les déplacements des pièces et les situations spéciales comme celles de l'échec ou du roque.

Examinons donc rapidement l'environnement et les règles associées aux Échecs afin d'identifier précisément les difficultés qu'elles posent :

* **les pions** : leur déplacement de base se fait en ligne droite et seulement vers l'avant. On retiendra deux exceptions de déplacement. La première s'applique lorsqu'il s'agit d'éliminer une pièce adverse, ce qui ne peut se faire que si cette dernière est située sur les premières cases respectives des diagonales de cases faisant face au pion bougé. La seconde se fait lors du premier mouvement de chaque pion, le joueur peut décider de le faire avancer d'une case ou de deux.

* **les fous :** ils ont la possibilité de se déplacer d'autant de cases qu'ils souhaitent en diagonale tant que le passage n'est pas obstrué par une autre pièce adverse ou non.

nale tant que le passage n'est pas obstrué par une autre pièce adverse ou non.

* **les cavaliers :** leur mouvement consiste à aller de deux cases dans une direction puis d'encore une autre case perpendiculairement à leur première trajectoire. Cela donne un trajet en forme de «L». La particularité des cavaliers est qu'ils ne sont pas bloqués par les autres pièces. La seule condition nécessaire à la réalisation de leur déplacement est qu'il n'y est pas d'allié sur la case ciblée. S'il s'agit d'un ennemi en revanche, le déplacement s'effectue et la pièce cible est éliminée.

* **les tours :** elles se déplacent verticalement et horizontalement d'autant de cases voulus tant qu'il n'y a pas de pièces pour les bloquer.

* **la dame :** la dame cumule les déplacements de la tour et du fou, elle peut aller en diagonale, à l'horizontale ou à la verticale sans limite de case mais est également bloquée par les autres pièces.

* **le roi :** il se meut case par case dans n'importe quelle direction. La seule condition quant à son déplacement est qu'il ne se mette pas en échec.

Les règles spéciales

* **le roque :** il s'agit d'un déplacement spécial réalisé avec la tour et le roi. L'objectif de ce rapport n'étant pas de détaillé précisément les règles du jeu d'échec, nous supposerons que le lecteur connaît les conditions de réalisation de ce mouvement.

* **le pat :** cela correspond à une situation dans laquelle aucun des deux joueurs ne peut effectuer de coup légal, c'est à dire que quelque soit la prochaine action de la part des deux camps le roi sera en échec et mat. On a alors match nul.

* **la promotion :** une fois qu'un pion atteint la dernière rangée lui faisant fasse, le joueur a obligation de le promouvoir, c'est à dire de le faire évoluer en n'importe quelle autre pièce.

* **la prise en passant :** c'est une autre situation particulière mettant en situation deux pions, là encore on supposera que le lecteur sait ce qu'il en ressort.

* **la protection du roi:** une pièce ne peut bouger si elle met en échec le roi.

L'environnement de jeu

* **le plateau :** il est ici question d'un plateau de 64 cases noires et blanches.

* **les camps/joueurs :** les Échecs font s'opposer deux joueurs dirigeant chacun un camp. Ils se différencient par leur couleur :noir ou blanc. Chaque joueur a droit à 50 coups maximum.

* **le positionnement :** en début de partie toutes les pièces de chaque joueur sont disposées de manière bien particulière.

Résumons donc ce que nous venons d'énoncer à l'aide de «boîtes» synthétisants les caractéristiques de chaque élément du jeu.

Pion
- déplacement - couleur - position de départ - prise en passant - promotion - protection du Roi

Fou
- déplacement - couleur - position de départ - protection du Roi - bloqué par les autres pièces

Cavalier
- déplacement - couleur - position de départ - protection du Roi - peut «sauter» par dessus les pièces

Tour
- déplacement - couleur - position de départ - protection du Roi - bloquée - roque

Dame
- déplacement - couleur - position de départ - protection du Roi - bloquée

Roi
- déplacement - couleur - position de départ - roque - bloqué - échec au Roi

Joueur 1
- couleur 1 - nombre de coups

Joueur 2
- couleur 2 - nombre de coups

Plateau
- 32 cases couleur 1 - 32 cases couleur 2 - 8 colonnes - 8 lignes

Voici donc les éléments qui vont nous être utiles dans l'organisation et la conception de notre code.

II - Solution technique et interaction entre les composants.

A) Définition des composants.

Il semble évident et légitime que la notion charnière qui va articuler le codage des pièces est celle de l'héritage. En effet, toutes les pièces ont pour caractéristiques communes une couleur, un numéro de ligne, de colonne, un nom et la possibilité de se déplacer ou non. Ainsi, nous avons établi une classe mère comportant tous ces attributs. La position de départ de chaque pièce en revanche n'a pas été indiqué dans cette classe. En effet, il est apparu plus commode d'attribuer une situation de début de jeu directement sur le plateau pour les positions de départ.

Nous obtenons donc cette première classe **Pièce.** Il s'agit d'une classe abstraite car nous avons la méthode deplacementPossible qui est une méthode virtuelle pure : nous avons vu précédemment que chaque type de pièce possède son déplacement propre.

Piece (abstrait)
#col : int #ligne : int #couleur : int #nom : char
+afficherPiece() : void +getNom() : void +getCouleur() : int +getLigne() : int +getCol() : int +setLigne(int) : void +setCol(int) : void +deplacementPossible(Plateau&,vector<Case*> &) : void (virtual)

C'est donc de cette classe mère que toutes les pièces vont hériter leurs attributs triviaux. Voyons maintenant au cas par cas les autres méthodes et attributs de chaque type de pièce.

Les pions, les tours, les fous, la dame et le roi seront simplement dotés d'un constructeur ainsi que d'une méthode définissant les déplacements possibles de chaque pièce. Il s'agit de la méthode virtuelle héritée de la classe Pièce. Précisons que dans le cas du Roi, la méthode incluera la possibilité de roquer.

Pion
+Pion (int, int, int, char) : pion +deplacementPossible(Plateau &, vector <Case*> &) : void

Fou
+Fou (int, int, int, char) : fou +deplacementPossible(Plateau &, vector <Case*> &) : void

Tour
+Tour (int, int, int, char) : tour +deplacementPossible(Plateau &, vector <Case*> &) : void

Dame
+Dame(int, int, int, char) : dame +deplacementPossible(Plateau &, vector <Case*> &) : void

Roi
+Roi(int, int, int, char) : roi +deplacementPossible(Plateau &, vector <Case*> &) : void

Nous avons ensuite le cavalier qui va disposer d'une méthode particulière. Rappelons qu'il est le seul à pouvoir «sauter» au dessus d'autres pièces et que son déplacement n'est dépendant que de l'occupation de la case qu'il cible. Il faut donc une fonction permettant de vérifier que si la destination du cavalier est occupée par un ennemi, le joueur peut réaliser le mouvement souhaité tandis que si c'est un allié il ne pourra pas.

Cavalier
+Cavalier(int, int, int, char) : cavalier +deplacementPossible(Plateau &, vector <Case*> &) : void +depValide(Plateau &, vector <Case*> &, int, int) : void

Maintenant que nous avons abordé toutes les pièces, intéressons nous à l'environnement de jeu. Nous savons que physiquement le début d'une partie d'échecs est marquée par la disposition des pièces sur leur case respective. Que serait un jeu d'échecs sans plateau ? Pas grand chose... Nous devons donc définir le support du jeu.

Notre plateau va être représenté par un tableau à deux dimensions composé de cases. Nous allons donc avoir deux classes : une pour le plateau et une pour les cases.

Case
#p : piece* #couleur : int #col : int #ligne : int
+case(int, int, int, Piece* = NULL) : case +setP (Piece* = NULL) : void +getP () : piece* +getLigne () : int +getCol () : int +getCouleur () : int +afficherCase () : void

Pour la classe **case** nous avons donc tout naturellement une couleur, un numéro de colonne et de ligne mais également un pointeur qui permet de déterminer la présence ou non d'une pièce sur la case en question. Les méthodes découlent donc logiquement, un constructeur, un setter et un getter pour la présence d'une pièce des getters pour la ligne, la colonne et la couleur et une méthode afficher.

Plateau
#plat[8][8] : case*
+Plateau () : plateau +void afficherPlateau () : void +getPlatIJ (int, int) : case* +getCaseIJ (int, int) : case

Dans le cas du jeu d'Échecs nous avons donc un plateau de 64 cases donc une matrice 8x8 ici représentée par un tableau à deux dimensions de pointeurs de case. Le constructeur va créer le plateau en positionnant toutes les pièces des deux joueurs. et nous avons ensuite les getters/-setters ainsi qu'une fonction d'affichage.

Maintenant que l'environnement a été représenté, il nous faut virtualiser le déroulement d'une partie. Nous allons donc définir trois nouvelles classes : joueur, partie et coup.

Joueur
#couleur : int #score : int
+Joueur (int, int) : joueur +getCouleur () : int +getScore () : int +setCouleur (int) : void +setScore (int) : void +demande_verif_creer_Coup (Plateau &, vector <Coup*> &) : void

Un joueur est caractérisé par sa couleur et par son score. Là encore les méthodes découlent naturellement : un constructeur, des getters/setters pour la couleur et le score, et une méthode pour vérifier que les coups qu'il exécute sont bien valides.

Partie
#Plateau P : plateau #coupsJoues : vector <Coup*> #blanchesPrises : vector <Piece*> #noiresPrises : vector <Piece*> #blanche : int #noire : int
+Partie () : partie +Menu () : void +jouerPartie (Joueur &, Joueur &) : void +afficherPrises () : void +rechercheRoi (Joueur &, Piece* &) : void +verif_etatPartie (Joueur &) : bool +verifPat (Joueur &) : bool +verifEchec (Joueur &) : bool +verifMat (Joueur &) : bool +enregistrerCoup () : void +restaurePartie (Joueur &, Joueur &) : void +annuleCoup () : void

Une classe un peu plus consistante que les autres. On va trouver dans les attributs le plateau, le nombre de pièces blanches prises, le nombre de noires prises, l'attribut «coupsJoues» va stocker tous les coups joués durant la partie et les attributs «blanche» et «noire» vont être des indicateurs permettant de savoir quelle est la couleur de la dernière pièce éliminée.

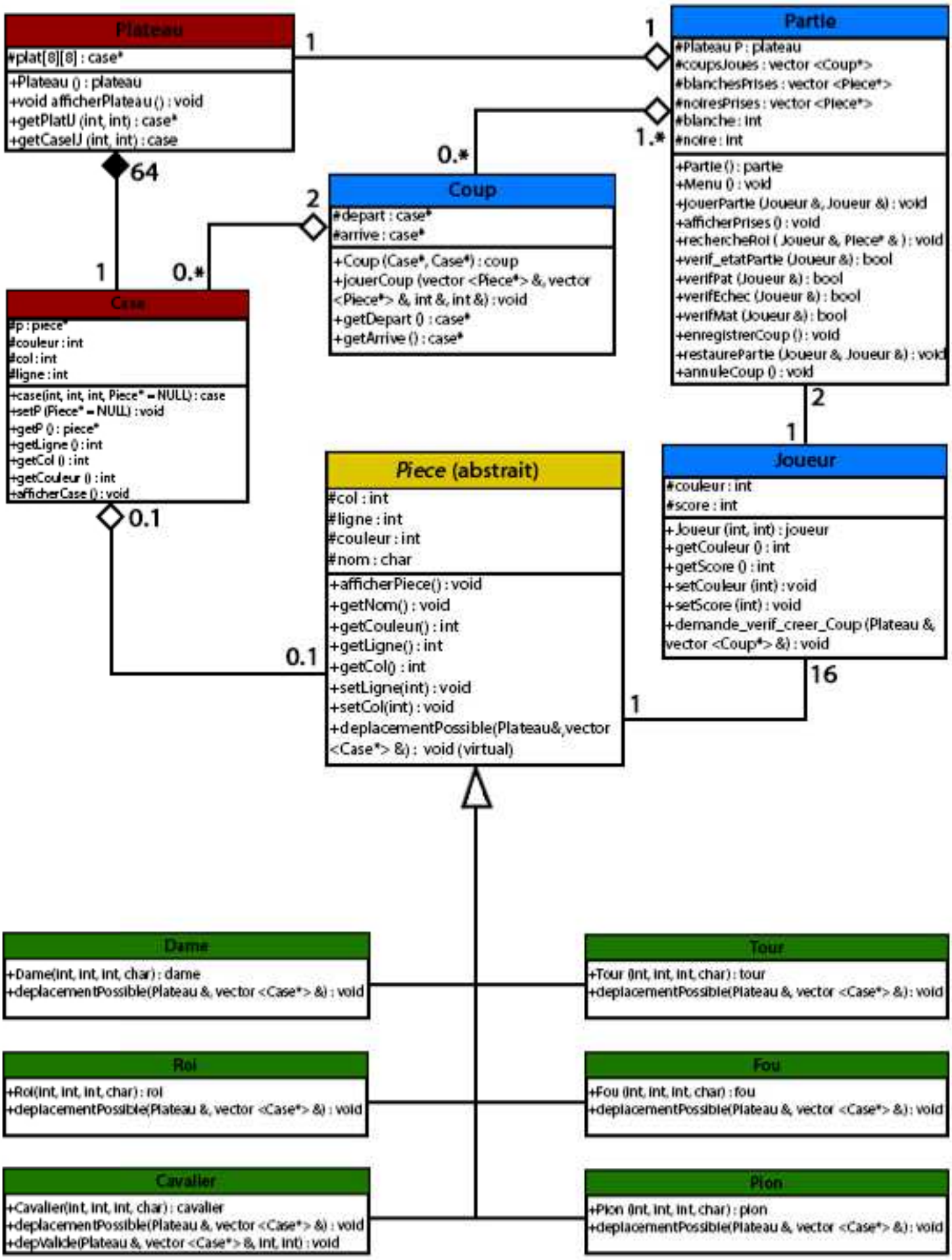
Pour ce qui est des méthodes nous avons donc tout d'abord un constructeur. La fonction «Menu» va permettre de laisser le choix à l'utilisateur entre faire une nouvelle partie ou bien en reprendre une sauvegardée (si partie sauvegardée il y a bien sûr). Dans le premer cas, nous avons la fonction «jouerPartie» qui va permettre de lancer une nouvelle partie, dans l'autre cas la fonction «restaurePartie» va s'exécuter et rétablir le plateau dans l'état dans lequel il était durant la dernière partie enregistrée. Nous avons ensuite une méthode qui permet d'afficher le nombre de chaque pièce de couleur pris, une fonction qui permet de parcourir l'échiquier afin de situer le roi pour ensuite activer selon la situation actuelle du plateau la méthode «verif_etat-Partie» qui selon la position du Roi va permettre de prévenir de l'état d'échec, d'échec et mat ou de pat en utilisant les trois fonctions «verifEchec», «verifMat» et «verifPat». L'utilisation de ces dernières font appel à la fonction «annuleCoup» : en effet, les vérifications sont faites en jouant tous les coups possibles du joueur pour vérifier qu'il est toujours en échec après, il faut donc ensuite annuler ces coups joués par le système. La dernière fonction permet d'enregistrer la partie en cours.

Coup
#depart : case* #arrive : case*
+Coup (Case*, Case*) : coup +jouerCoup (vector <Piece*> &, vector <Piece*> &, int &, int &) : void +getDepart () : case* +getArrive () : case*

La dernière classe utilisée est celle du Coup. Les attributs servent à identifier la case de départ et la case d'arrivée. Les méthodes : tout d'abord un constructeur. Puis une méthode d'exécution du coup qui va également vérifier si la case ciblée contient une pièce adverse ou non. Dans le cas où il y en aurait une, le système va l'éliminer et incrémenter les compteurs de pièces prises. Nous avons finalement des getters pour déterminer la prochaine case de départ de la pièce (qui correspond en fait à la case d'arrivée de ce tour) et vider la case de départ.

B) Intéraction entre les composants.

Sur la page suivante, nous allons voir comment intéragissent les composants entre eux à l'aide d'un diagramme UML.


Plateau
#plat[8][8] : case*
+Plateau () : plateau
+void afficherPlateau () : void
+getPlatIJ (int, int) : case*
+getCaseIJ (int, int) : case
1
64
Partie
#Plateau P : plateau
#coupsJoues : vector <Coup*>
#blanchesPrises : vector <Piece*>
#noiresPrises : vector <Piece*>
#blanche : int
#noire : int
+Partie () : partie
+Menu () : void
+jouerPartie (Joueur &, Joueur &) : void
+afficherPrises () : void
+rechercheRoi (Joueur &, Piece* &) : void
+verif_etatPartie (Joueur &) : bool
+verifPat (Joueur &) : bool
+verifEchec (Joueur &) : bool
+verifMat (Joueur &) : bool
+enregistrerCoup () : void
+restaurePartie (Joueur &, Joueur &) : void
+annuleCoup () : void
1
1.*
0.*
Coup
#depart : case*
#arrive : case*
+Coup (Case*, Case*) : coup
+jouerCoup (vector <Piece*> &, vector <Piece*> &, int &, int &) : void
+getDepart () : case*
+getArrive () : case*
2
1
0.*
Case
#p : piece*
#couleur : int
#col : int
#ligne : int
+case(int, int, int, Piece* = NULL) : case
+setP (Piece* = NULL) : void
+getP () : piece*
+getLigne () : int
+getCol () : int
+getCouleur () : int
+afficherCase () : void
2
1
Joueur
#couleur : int
#score : int
+Joueur (int, int) : joueur
+getCouleur () : int
+getScore () : int
+setCouleur (int) : void
+setScore (int) : void
+demande_verif_creer_Coup (Plateau &, vector <Coup*> &) : void
0.1
0.1
Piece (abstrait)
#col : int
#ligne : int
#couleur : int
#nom : char
+afficherPiece() : void
+getNom() : void
+getCouleur() : int
+getLigne() : int
+getCol() : int
+setLigne(int) : void
+setCol(int) : void
+deplacementPossible(Plateau&,vector <Case*> &) : void (virtual)
16
1
Dame
+Dame(int, int, int, char) : dame
+deplacementPossible(Plateau &, vector <Case*> &) : void
Tour
+Tour (int, int, int, char) : tour
+deplacementPossible(Plateau &, vector <Case*> &) : void
Roi
+Roi(int, int, int, char) : roi
+deplacementPossible(Plateau &, vector <Case*> &) : void
Fou
+Fou (int, int, int, char) : fou
+deplacementPossible(Plateau &, vector <Case*> &) : void
Cavalier
+Cavalier(int, int, int, char) : cavalier
+deplacementPossible(Plateau &, vector <Case*> &) : void
+depValide(Plateau &, vector <Case*> &, int, int) : void
Pion
+Pion (int, int, int, char) : pion
+deplacementPossible(Plateau &, vector <Case*> &) : void

III - Bilan.

A) Manuel d'utilisation.

Il s'agit malheureusement d'un software très simpliste qui ne possède pas d'interface graphique. Son utilisation va donc se faire directement en console. Après exécution du fichier main dans un terminal nous avons donc le système qui affiche la chose suivante :

```
Pour commencer une nouvelle partie entrez 1, pour reprendre une partie
enregistrée entrez 2.
```

On tape donc 1 pour commencer une nouvelle partie ou 2 si l'on désire reprendre une partie déja commencée car le jeu dispose d'une sauvegarde automatique a chaque tour (donc on peut quitter le jeu a tout moment la partie sera sauvegardée).

Par la suite nous allons considérer qu'on a lancé une nouvelle partie. L'échiquier s'affiche donc ainsi :

```
0   T  C  F  D  R  F  C  T
1   P  P  P  P  P  P  P  P
2   -  -  -  -  -  -  -  -
3   -  -  -  -  -  -  -  -
4   -  -  -  -  -  -  -  -
5   -  -  -  -  -  -  -  -
6   p  p  p  p  p  p  p  p
7   t  c  f  d  r  f  c  t
```

Pour jouer, c'est très simple : le système va demander successivement le numéro de colonne et de le ligne de la pièce qu'on souhaite déplacer, puis les numéros de la case sur laquelle on veut aller.

```
Entrez la colonne de la pièce à déplacer :
4
Entrez la ligne de la pièce à déplacer :
6
Entrez la colonne où vous souhaitez déplacer la pièce :
4
Entrez la ligne où vous souhaitez déplacer la pièce :
4
```

Si les coordonnées indiquées pour la pièce choisie ne sont pas sur l'échiquier, si elle n'appartient pas au joueur entrain d'entrer les commandes ou si c'est une piece lui appartenant mais qui n'a pas de déplacement possible, le système va lui redemander des coordonnées.

De même pour les coordonnées de destination, si elles sont fausses le jeu indique au joueur les cases que la pièces sélectionnée peut atteindre et redemande ensuite les coordonnées de destination. Cependant le jeu indique un coup qui peut mettre ou laisser le roi du joueur en echec, mais lorsqu'il valide son choix, le système lui signale que le coup le met en echec et lui fait rechoisir une autre pièce à déplacer.

Si un joueur se retrouve en échec, le jeu l'indique. S'il est en cas de pat ou d'échec et mat, c'est la fin de la partie.

À chaque tour s'affichent l'échiquier mis a jour ainsi que les pièces prises.

Par exemple avec le coup précédent, on aura d'afficher la figure suivante :

```
    0  1  2  3  4  5  6  7

0   T  C  F  D  R  F  C  T
1   P  P  P  P  P  P  P  P
2   -  -  -  -  -  -  -  -
3   -  -  -  -  -  -  -  -
4   -  -  -  -  p  -  -  -
5   -  -  -  -  -  -  -  -
6   P  P  P  P  -  P  P  P
7   t  c  f  d  r  f  c  t
1
Pièces blanches prises :

Pièces noires prises :
```

Voilà tout, il ne reste plus qu'à vous trouver un adversaire et à jouer !

B) Conclusion personnelle.

Même si le temps a manqué pour perfectionner la chose notamment au niveau de l'interface graphique, le projet s'est révélé on ne peut plus intéressant pour plusieurs raisons. La première d'entre elles est que la conception du jeu force une imbrication de classes rigoureuse mais permissive, ce qui laisse une assez grande liberté dans les possibilités de conception. La seconde est que les échecs est un jeu simple mais disposant de règles spéciales qu'il est important de gérer. La polyvalence des situations possibles nous a permis de mettre au point des solutions différentes et nous a obligés à trouver des voies optimales pour développer les réponses

aux problèmes rencontrés sans en générer d'autres. Notre projet final n'est donc pas encore achevé à 100%, mais nous nous attellerons sans doute à le finir dans l'optique de progresser et d'assouvir une certaine satisfaction du travail accompli.

Fin du projet...

... échec et mat !

Remerciements

Nous souhaitons remercier les personnes suivantes pour leur participation directe ou indirecte à l'ellaboration de notre projet.

Pour leur aide durant les TP, leurs cours et leurs directives, merci à :

Mr Christophe JAILLET
Mr Stanislas RENARD
Mr Philippe BAUDOUIN
Mme Itheri Yahiahoui

Merci également aux sites suivants et à leur communauté pour le contenu et l'aide qu'ils nous ont apportés :

http://www.cplusplus.com/
http://www.developpez.com/
http://fr.openclassrooms.com/